Pyspark interview questions for freshers
Source 1:
https://www.knowledgehut.com/interview-questions/pyspark-interview-questions#collapse-beginner-9175
Source 2:-
https://www.interviewbit.com/pyspark-interview-questions/
Github code repository:-
https://github.com/spark-examples/pyspark-examples
https://github.com/edyoda/pyspark-tutorial
https://github.com/hyunjoonbok/PySpark

LEVEL EASY:

1. **Explain the startsWith() and endsWith() methods.**

In PySpark, startsWith() and endsWith() methods come in the Column class. These methods are used to search DataFrame rows by checking if the column value starts or ends with a specific value. Both are used to filter data in our application.

- startsWith() method: This method is used to return a boolean value. It indicates TRUE if the value in the column starts with the specified string, and FALSE if the value in that column does not match.
- endsWith() method: This method is used to return a boolean value. It indicates TRUE if the column value ends with the specified string, and FALSE if the match for that column value is not satisfied. Both methods are case sensitive.

2. **Is PySpark faster than Pandas?**

   Yes, PySpark is faster than Pandas. Because it supports parallel execution of statements in a distributed environment. For example, PySpark can run on different cores and machines, which is not available with Pandas. This is the main reason PySpark is faster than Pandas.

3. **What is the use of PySpark StorageLevel?**

   PySpark StorageLevel is used to control RDD storage. It can control how and where the RDD is stored. The PySpark StorageLevel decides whether the RDD is stored in memory, on disk, or both. It also determines if we need to replicate the RDD partitions or serialize the RDD. The code for PySpark StorageLevel looks like:

```
class PySpark.StorageLevel( useDisk, useMemory, useOfHeap, deserialized, replication = 1)
```

4. **Which are the most frequently used Spark ecosystems?**

There are some most used Spark ecosystems like Spark SQL, Spark Streaming, GraphX, MLlib, SparkR, etc.

- Spark SQL is for developers. It is also known as Shark.
- Spark Streaming is used for processing live data streams.
- Graphx is used for generating and calculating graphs.
- MLlib (also called as Machine Learning Algorithms)
- SparkR is used to promote the R programming language in the Spark engine.

5. What is PySpark?

PySpark is an Apache Spark interface in Python. It is used for collaborating with Spark using APIs written in Python. It also supports Spark's features like Spark DataFrame, Spark SQL, Spark Streaming, Spark MLlib and Spark Core. It provides an interactive PySpark shell to analyse structured and semi-structured data in a distributed environment. PySpark supports reading data from multiple sources and different formats. It also facilitates the use of RDDs (Resilient Distributed Datasets). PySpark features are implemented in the py4j library in python.

6. What are the characteristics of PySpark?

Abstracted Nodes: This means that the individual worker nodes can not be addressed.
Spark API: PySpark provides APIs for utilising Spark features.

Map-Reduce Model: PySpark is based on Hadoop’s Map-Reduce model; this means that the programmer provides the map and the reduce functions.
Abstracted Network: Networks are abstracted in PySpark which means that the only possible communication is implicit communication.

7. **What are the advantages and disadvantages of PySpark?**

Advantages of PySpark:

- Simple to use: Parallelized code can be written in a simpler manner.
- Error Handling: PySpark framework easily handles errors.
- Inbuilt Algorithms: PySpark provides many of the useful algorithms in Machine Learning or Graphs.
- Library Support: Compared to Scala, Python has a huge library collection for working in the field of data science and data visualisation.
- Easy to Learn: PySpark is an easy to learn language.

Disadvantages of PySpark:

- Sometimes, it becomes difficult to express problems using the MapReduce model.
- Since Spark was originally developed in Scala, while using PySpark in Python programs they are relatively less efficient and approximately 10x times slower than the Scala programs. This would impact the performance of heavy data processing applications.
- The Spark Streaming API in PySpark is not mature when compared to Scala. It still requires improvements.
- PySpark cannot be used for modifying the internal function of the Spark due to the abstractions provided. In such cases, Scala is preferred.

8. **What are the prerequisites to learn PySpark?**

PySpark framework is easy to learn and implement. No programming language or database experience is required to learn PySpark. It is very easy to learn if you know programming languages and frameworks. Before getting familiar with PySpark concepts, you should have some knowledge of Apache Spark and Python. Learning advanced concepts of PySpark is very helpful.

9. **How can you implement machine learning in Spark?**

   MLlib can be used to implement machine learning in Spark. Spark provides a scalable machine learning dataset called MLlib. It is primarily used to make machine learning scalable and lightweight, with common learning algorithms and use cases such as clustering, decay filtering, and dimensionality reduction. This is how machine learning can be implemented in Spark.

10. **Tell me the difference between PySpark and other programming languages.**
    There are some basic differences between PySpark and other programming languages. PySpark has its built-in APIs, but other programming languages require APIs to be integrated externally from third parties. Next difference is, implicit communications can be done in PySpark, but it is not possible in other programming languages. PySpark is map based, so developers can use maps to reduce functions. PySpark allows for multiple nodes, again which is not possible in other programming languages.

11. **Can we use PySpark in the small data set?**
    No, we cannot use PySpark with small data sets. They are not very useful as they are typical library systems with more complex objects than accessible objects. PySpark is great for large amounts of records, so it should be used for large data sets.

12. **What is your thought on PySpark is important in data science?**
    Data Science is based on two programming languages, Python and Machine Learning. PySpark is integrated with Python. There are interfaces and built-in environments which are made using both Python and Machine Learning. This makes PySpark an essential tool for data science. Once the dataset is processed, the prototype model is transformed into a production-ready workflow. Because of such reasons, PySpark is important in data science .

13. **What are the main file systems that are supported by Spark?**

Spark can form distributed datasets from any storage source supported by different file systems. These file systems include:

- Hadoop Distributed File Systems (HDFS)
- Local File System

- Cassandra
- HBase
- Amazon S3

Also, Spark supports text files, Sequence files, and any other Hadoop Input Format.

14. **What is RDD in PySpark?**
    In PySpark, the full form of RDD is Resilient Distributed Datasets. This RDD is the central data structure of PySpark. RDD is a low-level object that is very efficient at performing distributed tasks. RDDs are elements that can be run and manipulated on multiple nodes to perform parallel processing within a cluster. These are immutable items. This means that once the RDD is created it can not be changed. Also, RDDs are fault tolerant. In case of failure, they are automatically restored. Multiple operations can be applied to an RDD to accomplish a specific task. You can learn more about this concept at Certified Programming courses.

15. **What can you tell me about PySpark SparkContext?**

    SparkContext serves as the entry point for all Spark functions. When the Spark application runs, the driver program is started and the main function and SparkContext are started. The driver program then executes the operation within the worker node's executor. In PySpark, SparkContext is recognized as PySpark SparkContext. Using Py4J library, it starts a JVM and creates a JavaSparkContext. PySpark SparkContext is available as 'sc' by default, so it does not create a new SparkContext.
16. **What is PySpark SparkFiles?**
    PySpark SparkFiles is used to load files into Apache Spark applications. This is one of the functions under SparkContext that you can call with sc.addFile() to load files into Apache Spark. You can also use SparkFiles to get the path using SparkFile.get(). It can also be used to resolve paths to files added using the sc.addFile() method. The class methods in the SparkFiles directory are getrootdirectory() and get(filename).

What do you know about PySpark Serializers?

In PySpark, serialization is the process used to perform performance tuning in Spark. PySpark supports serializers because it needs to continuously inspect data sent and received to disk or storage over the network. There are two types of serializers that PySpark supports. The serializers are:

- **PickleSerializer:** This is used to serialize objects with its PickleSerializer class in Python using the PySpark. This serializer supports all Python objects.
- **MarshalSerializer:** The MarshalSerializer is used to perform object serialization. This is available using the PySpark MarshalSerializer class. This serializer is much faster than the PickleSerializer, but it supports limited types.

17. **What is PySpark ArrayType? Explain with an example.**

PySpark ArrayType is a collection data type that extends PySparks's DataType class. And this DataType class is the superclass of all types. A PySpark ArrayType only contains elements of the same type. We can also create an instance of ArrayType using the ArrayType() method. This method accepts two arguments. The arguments are valueType and valueContainsNull.

1. valueType: This argument must extend PySpark's DataType class.
2. valueContainsNull: This is an optional argument. It indicates whether the value can accept nulls and it is set to True by default.

For example:

```
from PySpark.sql.types import StringType, ArrayType

ArrayColumn = ArrayType(StringType().False)
```

18. **What do you understand by PySpark DataFrames?**
    A PySpark DataFrame is a distributed collection of well-organized data. These are just like relational database tables, arranged in named columns. PySpark DataFrames are more optimized than the R or Python programming languages as they can be created from a variety of sources such as Hive tables, structured data files, existing RDDs and external databases.

The biggest advantage of PySpark DataFrame is that data in PySpark DataFrame is distributed to different computers in the cluster and operations performed on it are executed in parallel on all computers. This makes it easy to handle large collections of petabytes of structured or semi-structured data.

What are the advantages of PySpark RDD?

PySpark RDDs have the following advantages:

- In-Memory Processing: PySpark's RDD helps in loading data from the disk to the memory. The RDDs can even be persisted in the memory for reusing the computations.
- Immutability: The RDDs are immutable which means that once created, they cannot be modified. While applying any transformation operations on the RDDs, a new RDD would be created.
- Fault Tolerance: The RDDs are fault-tolerant. This means that whenever an operation fails, the data gets automatically reloaded from other available partitions. This results in seamless execution of the PySpark applications.
- Lazy Evolution: The PySpark transformation operations are not performed as soon as they are encountered. The operations would be stored in the DAG and are evaluated once it finds the first RDD action.
- Partitioning: Whenever RDD is created from any data, the elements in the RDD are partitioned to the cores available by default

19. What are the different types of algorithms supported in PySpark?

Different types of algorithms supported in PySpark are:

- spark.mllib
- mllib.regression
- mllib.recommendation
- mllib.clustering

- mllib.classification
- mllib.linalg
- mllib.fpm

20. What is PySpark UDF?

UDF stands for User Defined Functions. In PySpark, UDF can be created by creating a python function and wrapping it with PySpark SQL's udf() method and using it on the DataFrame or SQL. These are generally created when we do not have the functionalities supported in PySpark's library and we have to use our own logic on the data. UDFs can be reused on any number of SQL expressions or DataFrames.

21. What is PySpark SparkConf?

PySpark SparkConf is mainly used if we have to set a few configurations and parameters to run a Spark application on the local/cluster. In other words, we can say that PySpark SparkConf is used to provide configurations to run a Spark application.

22. What is SparkCore, and what are the key functions of SparkCore?

SparkCore is a general execution engine for the Spark platform, including all the functionalities. It offers in-memory computing capabilities to deliver a good speed, a generalized execution model to support various applications, and Java, Scala, and Python APIs that make the development easy.

The main responsibility of SparkCore is to perform all the basic I/O functions, scheduling, monitoring, etc. It is also responsible for fault recovery and effective memory management.

The key functions of SparkCore are:

- Perform all the basic I/O functions
- Job scheduling
- Monitoring jobs
- Memory management

  - Fault-tolerance
  - Interaction with storage systems

23. What do you understand about Spark drivers?

The Spark driver is a plan that runs on the master node of a machine. It is mainly used to state actions and alterations on data RDDs.

24. What is PySpark SparkJobinfo?

The PySpark SparkJobinfo is used to get information about the SparkJobs that are in execution.

Following is the code for using the SparkJobInfo:

1. class SparkJobInfo(namedtuple("SparkJobInfo", "jobId stageIds status ")):

27. What is the use of Akka in PySpark?

Akka is used in PySpark for scheduling. When a worker requests a task to the master after registering, the master assigns a task to him. In this case, Akka sends and receives messages between the workers and masters.

28. What do you understand about RDD Lineage?

The RDD lineage is a procedure that is used to reconstruct the lost data partitions. The Spark does not hold up data replication in the memory. If any data is lost, we have to rebuild it using RDD lineage. This is the best use case as RDD always remembers how to construct from other datasets.

LEVEL MEDIUM:

1. **What do you understand by custom profilers in PySpark?**

PySpark supports custom profilers. Custom profilers are used to build predictive models. A profiler is also used to ensure that the data is valid and can be used at the time of consumption. If we want a custom profiler, we will need to define some methods. These methods are:

1. stats: This method is used to return aggregated profiling statistics.
2. profile: This method is used to create a kind of system profile.
3. dump: This is used to back up the profile to the specified path.
4. dump(id, path): This is used to dump a specific RDD ID to the specified path.
5. add: This is used to add a profile to an existing cumulative profile. A profile class must be selected when creating a SparkContext.

2. **What do you mean by cluster manager? What are the different cluster manager types that are supported by PySpark?**

In PySpark, the cluster manager is a cluster-mode platform that facilitates Spark execution by allotting all resources worker nodes as needed.

The Spark Cluster Manager ecosystem includes a master node and multiple worker nodes. Master nodes, with the help of the cluster manager, provide resources, such as memory and processor allocation, to worker nodes according to the node's needs.

PySpark supports different cluster managers. Those can be explained as:

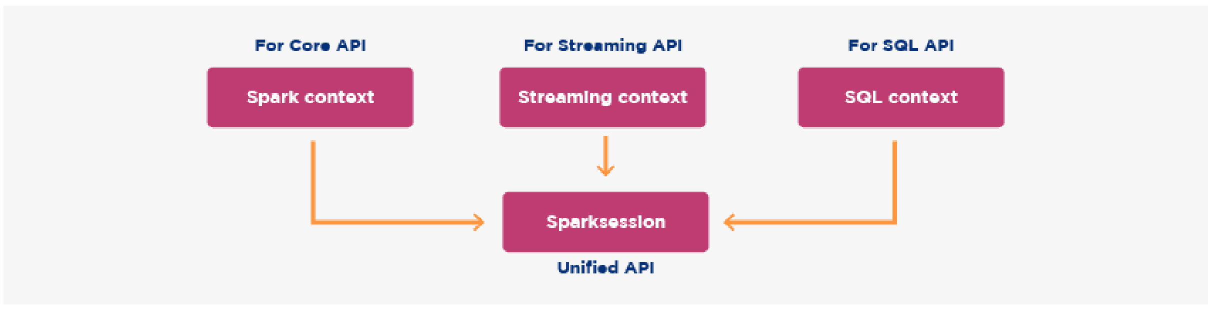
- Standalone: Standalone is a very simple cluster manager that comes with Spark.
- Apache Mesos: This cluster manager is useful to run Hadoop MapReduce and PySpark apps.
- Hadoop YARN: This cluster manager is used with Hadoop2.
- Kubernetes: This is an open-source cluster manager that helps to automate the deployment, scaling, and automated management of containerized apps.

- Local: This cluster manager is a path for running Spark applications on laptops/desktops.

3. **What is SparkSession in PySpark?**

   In PySpark, SparkSession is the application entry point. For the first version of PySpark, it used SparkContext as an entry point. SparkSession replaces the SparkContext since PySpark version 2.0. From the PySpark version 2.0, SparkSession serves as a starting point to access all PySpark functions related to RDDs, DataFrames, Datasets, etc. This is also the unified API used to access SQLContext, StreamingContext, HiveContext, and all other contexts within it to replace PySpark.

   SparkSession internally makes SparkContext and SparkConfig according to details provided in SparkSession. We can also create a SparkSession using the builder patterns.



4. **What is the common workflow of Spark program?**

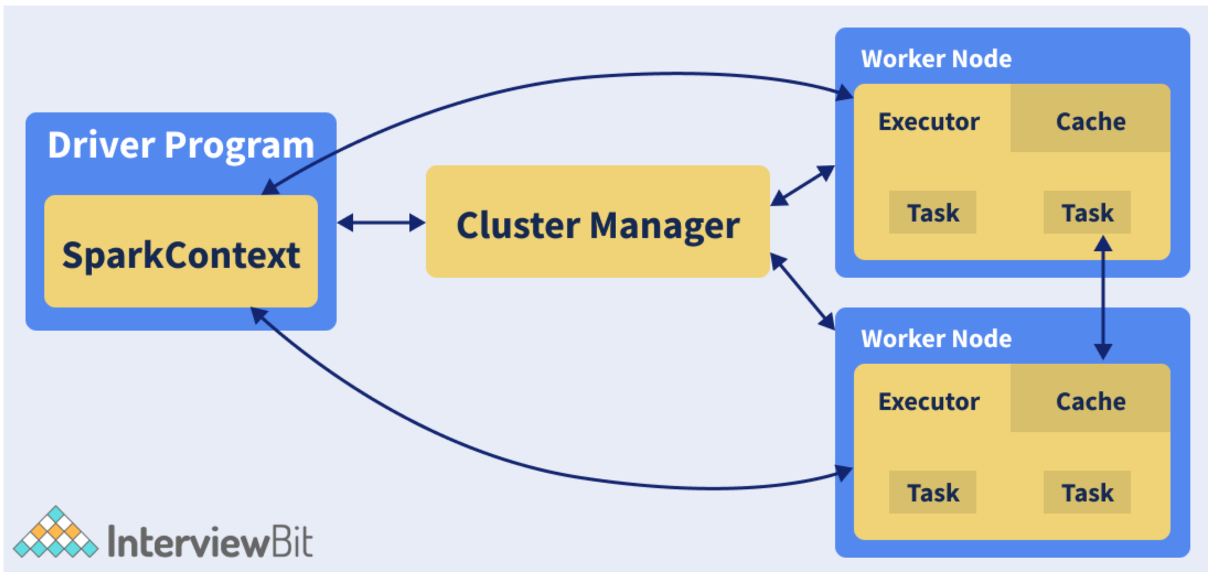
   There is a common workflow followed by a spark program. The first step is to create the input RDD according to the external data. Data can come from a variety of data sources. Next, after the RDD is created, depending on business logic, RDD transformation operations such as filter() and map() are performed to create a new RDD. In case, if we need to reuse the intermediate RDDs for later purposes, we can keep them. Finally, if there are action operations like first(), count(), etc., Spark fires them up to start parallel computation. This workflow is used by Spark program.

5. .Why are Partitions immutable in PySpark?

In PySpark, every transformation generates a new partition. Partitions use HDFS API to make partitions immutable, distributed, and fault-tolerant. Partitions are also aware of data locality

6. **What is PySpark Architecture?**

   PySpark similar to Apache Spark works in master-slave architecture pattern. Here, the master node is called the Driver and the slave nodes are called the workers. When a Spark application is run, the Spark Driver creates SparkContext which acts as an entry point to the spark application. All the operations are executed on the worker nodes. The resources required for executing the operations on the worker nodes are managed by the Cluster Managers. The following diagram illustrates the architecture described:



7. What is the Parquet file in PySpark?

In PySpark, the Parquet file is a column-type format supported by several data processing systems. By using the Parquet file, Spark SQL can perform both read and write operations.

The Parquet file contains a column type format storage which provides the following advantages:

- It is small and consumes less space.

- It facilitates us to fetch specific columns for access.
- It follows type-specific encoding.
- It offers better-summarized data.
- It contains very limited I/O operations.

8. What do you understand by "joins" in PySpark DataFrame? What are the different types of joins available in PySpark?

In PySpark, joins merge or join two DataFrames together. It facilitates us to link two or multiple DataFrames together.

INNER Join, LEFT OUTER Join, RIGHT OUTER Join, LEFT ANTI Join, LEFT SEMI Join, CROSS Join, and SELF Join are among the SQL join types PySpark supports. Following is the syntax of PySpark Join.

Syntax:

1. join(self, other, on=None, how=None)

Parameter Explanation:

The join() procedure accepts the following parameters and returns a DataFrame:

- **"other":** It specifies the join's right side.
- **"on":** It specifies the join column's name.
- **"how":** It is used to specify an option. Options are inner, cross, outer, full, full outer, left, left outer, right, right outer, left semi, and left anti. The default is inner.

Types of Join in PySpark DataFrame

Join String	Equivalent SQL Join
inner	INNER JOIN
outer, full, fullouter, full_outer	FULL OUTER JOIN
left, leftouter, left_outer	LEFT JOIN
right, rightouter, right_outer	RIGHT JOIN
cross	
anti, leftanti, left_anti	
semi, leftsemi, left_semi	

9. What is PySpark Partition? How many partitions can you make in PySpark?

PySpark Partition is a method of splitting a large dataset into smaller datasets based on one or more partition keys. It enhances the execution speed as transformations on partitioned data run quicker because each partition's transformations are executed in parallel. It also facilitates us to create a partition on multiple columns using partitionBy() by passing the columns you want to partition as an argument to this method.

Syntax:

1. partitionBy(self, *cols)

In PySpark, it is recommended to have 4x of partitions to the number of cores in the cluster available for application.

10. What are the main attributes used in SparkConf?

Following is the list of main attributes used in SparkConf:

- **set(key, value):** This attribute is used for setting the configuration property.
- **setSparkHome(value):** This attribute enables the setting Spark installation path on worker nodes.
- **setAppName(value):** This attribute is used for setting the application name.
- **setMaster(value):** This attribute is used to set the master URL.
- **get(key, defaultValue=None):** This attribute supports getting a configuration value of a key.

11. How can we trigger automatic cleanups in Spark to handle accumulated metadata?

We can trigger the automatic cleanups in Spark by setting the parameter ' Spark.cleaner.ttl' or separating the long-running jobs into dissimilar batches and writing the mediator results to the disk.

12. What is the difference between get(filename) and getrootdictonary()?

The main difference between get(filename) and getrootdirectory() is that get(filename) is used to get the correct path to the file that is added using SparkContext.addFile(). On the other hand, getrootdirectory() is used to get the root directory containing the file added using SparkContext.addFile().

13. **What is the use of Spark execution engine?**

The Apache Spark execution engine is a graph execution engine that makes it easy for users to explore large datasets with high presentation. If we want the data to be manipulated at different stages of processing, we need to keep Spark in memory for a radical performance boost.

14. **How is Spark SQL different from HQL and SQL?**

Hive is used in HQL (Hive Query Language), and Spark SQL is used in Structured Query Language to process and query data. We can easily connect SQL table and HQL table to Spark SQL. Flash SQL is used as a unique segment on the Spark Core engine that supports SQL and Hive Query Language without changing the sentence structure.

LEVEL HARD:

1. What are the types of PySpark's shared variables and why are they useful?

Whenever PySpark performs a transform operation using filter(), map(), or reduce(), they are executed on a remote node that uses the variables supplied with the tasks. These variables are not reusable and cannot be shared across jobs because they are not returned to the controller. To solve the problem of reusability and sharing, we use shared variables in PySpark. There are two types of shared variables:

Broadcast Variables:

They are also known as read-only shared variables. These are used in cases of data lookup requests. These variables are cached and made available on all cluster nodes to use. These variables are not sent with every task. Instead, they are distributed to nodes using efficient algorithms to reduce communication costs. When we run an RDD task operation that uses Broadcast variables, PySpark does this:

- The task is divided into different phases with distributed mixing. Actions are performed in these phases.
- The phases are then divided into tasks.
- Broadcast variables are broadcast to jobs if jobs need to use them.

Broadcast variables are created in PySpark using the broadcast(variable) method of the SparkContext class. The main reason for the use of broadcast variables is that the variables are not sent to the tasks when the broadcast function is called. They will be sent when the variables are first required by the executors.

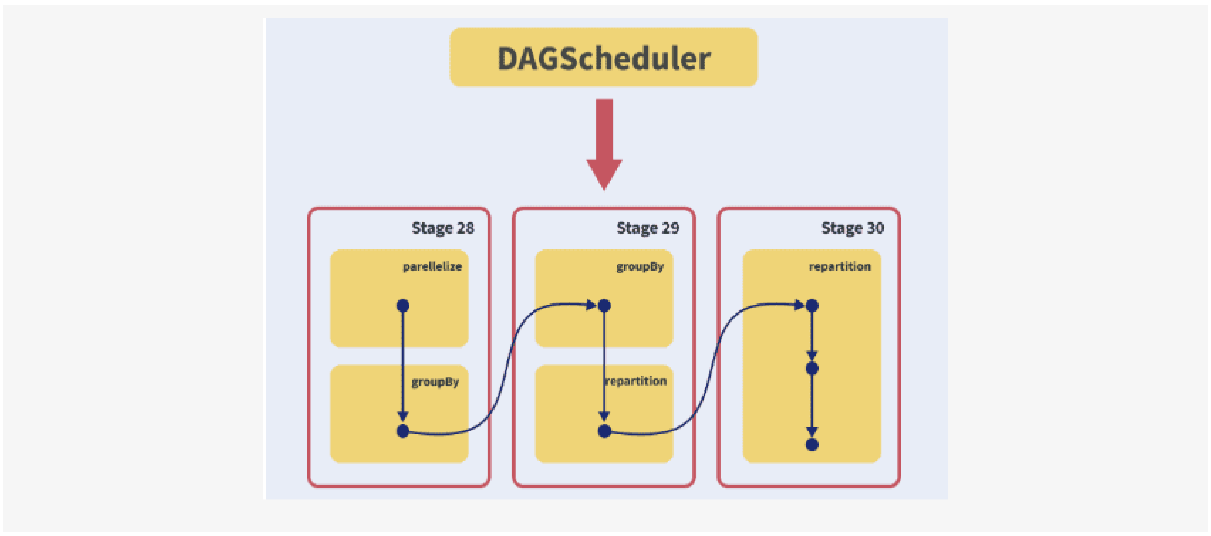
Accumulator Variables:

These variables are known as updatable shared variables. These variables are added through associative and commutative methods. They are used for performing counter or sum operations. PySpark supports the default creation of numeric type accumulators. It can also be used to add custom accumulator types. The custom types are of two types: Named accumulator and Unnamed accumulator.

- Named Accumulator: These accumulators are seen under the “Accumulator” tab in the PySpark web UI.
- Unnamed Accumulator: These accumulators are not seen under the “Accumulator” tab in the PySpark web UI.

2. What is DAG Scheduler in pyspark ?

One of the most frequently posed PySpark Interview Questions, be ready for it. The DAG is a full form of Direct Acyclic Graph. In Spark, DAGScheduler represents scheduling layer that implements task scheduling in a phase-oriented manner using tasks and phases. The logical execution plan (the dependencies of the transformation action line on the RDD) is transformed into a physical execution plan consisting of phases. It calculates the DAG of phases needed for each task and keeps track of which phases are realized RDDs and finds the minimum schedule for executing the tasks. These stages are then sent to the TaskScheduler to run the stages. This is shown in the image below:



DAGScheduler computes the DAG execution for the task. It specifies preferred locations for running each task. It handles failure due to loss of output files during mixing.

PySpark DAGScheduler follows an event queue architecture. Here, the thread publishes events of type DAGSchedulerEvent, such as a new stage or task. DAGScheduler then reads the phases and executes them sequentially in topological order.

3. What are the different methods for creating RDD in PySpark?

In PySpark, there are various methods used to create RDD. Let's discuss them one by one.

Using sparkContext.parallelize() - SparkContext's parallelize() method can be used to create RDDs. This method retrieves an existing collection from the controller and parallelizes it. This is the basic approach to creating an RDD and is used when we have data already present in memory. This also requires all data to be present in the driver before the RDD is created. The code to create an RDD using the parallelize method for the python list is:

```
list = [1,2,3,4,5,6,7,8,9,10,11]

rdd = spark.sparkContext.parallelize(list)
```

Using sparkContext.textFile() - We can read .txt file and convert it into RDD. The syntax looks like:

```
rdd_txt = spark.sparkContext.textFile("/path/to/textFile.txt")
```

Using sparkContext.wholeTextFiles() - This method returns PairRDD. PairRDD is an RDD that contains key-value pairs. In this PairRDD, file path is the key and file content is the value. This method reads entire file into an RDD as a single record. Besides, this method can also read files in other formats like CSV, JSON, parquet, etc. and create the RDDs.

```
rdd_whole_text = spark.sparkContext.wholeTextFiles("/path/to/textFile.txt")
```

Empty RDD with no partition using sparkContext.emptyRDD - The RDD without any data is known as empty RDD. We can make such RDDs which have no partitions by using the emptyRDD() method. The code piece for that will look like:

empty_rdd = spark.sparkContext.emptyRDD

empty_rdd_string = spark.sparkContext.emptyRDD[String]

Empty RDD with partitions using sparkContext.parallelize - When we require partition but not the data, then we create empty RDD by using the parallelize method. For example, below code create the empty RDD with 10 partitions:

empty_partitioned_rdd = spark.sparkContext.parallelize([],10)

4. How will you create PySpark UDF?

Let’s understand the process through an example. Here, we want to capitalize the first letter of every word in a string. There is no default feature in PySpark that can achieve this. However, we can make this happen by creating a UDF capitalizeWord(str) and using it on the DataFrames.

First, we will create a Python function capitalizeWord(). This function takes a string as an input and then capitalizes the first character of every word.

def capitalizeWord(str):

result = “”

words = str.split(“”)

for word in words:

```
result = result + word[0:1].upper() + word[1:len(x)] + “”
```

```
return result
```

Now, we will register the function as a PySpark UDF using the udf() method from org.apache.spark.sql.functions.udf package. This package must be imported. This function will return the object of class org.apache.spark.sql.expressions.UserDefinedFunction. The code for converting a function to UDF is:

```
capitalizeWordUDF = udf(lambda z: capitalizeWord(z), StringType())
```

Next step is to use UDF with DataFrame. We can apply UDF on a Python DataFrame as it will act as the built-in function of DataFrame. Consider we have a DataFrame stored in variable df, which has the columns as ID_COLUMN and NAME_COLUMN. Now, to capitalize every first character of the word, we will code as:

```
df.select(col(“ID_COLUMN”), convertUDF(col(“NAME_COLUMN”))

.alias(“NAME_COLUMN”) )

.show(truncate = False)
```

UDFs must be designed so that the algorithms are efficient and take up less time and space. If care is not taken, the performance of DataFrame operations will be affected.

5. What is PySpark SQL? Explain with an example.

PySpark SQL is the most famous PySpark module used to process structured columnar data. Once the DataFrame is created, we can work with the data using SQL syntax. Spark SQL is used to pass native raw SQL queries to Spark using select, where group by, join, union, etc. To use PySpark SQL, the first step is to create a temporary table on a DataFrame using the createOrReplaceTempView() method. Once created, the table is accessible within a SparkSession using the sql() function. When the SparkSession is terminated, the temporary table will get dropped.

For example, consider that we have a DataFrame assigned to the variable df which contains Name, Age and Gender of Students as the columns. Now, we will create a temporary table of the DataFrame that gets access to the SparkSession by using the sql() function. The SQL queries can be run within the function.

```
df.createOrReplaceTempView("STUDENTS")

df_new = spark.sql("SELECT * from STUDENTS")

df_new.printSchema()
```

The schema will be shown as:

```
>> df.printSchema()

root

|-- Name: string (nullable = true)

|-- Age: integer (nullable = true)

|-- Gender: string (nullable = true)
```

6. How to inner join two DataFrames?

We can use join() method that is present in PySpark SQL. The syntax of the method looks like:

```
join(self, other, on = None, how = None)

emp_dept_df = empDF.join(deptDF,empDF.empdept_id==deptDF.dept_id,"inner").show(truncate = False
```

6. What is PySpark Streaming? How can you stream data using TCP/IP protocol?

```
from PySpark import SparkContext
from PySpark.streaming

import StreamingContext from PySpark.sql

import SQLContext from PySpark.sql.functions

import desc

sc = SparkContext()

ssc = StreamingContext(sc, 10)

sqlContext = SQLContext(sc)

socket_stream = ssc.socketTextStream("127.0.0.1", 5555)

lines = socket_stream.window(20) df.printSchema()
```

7. Explain different levels of persistence that exist in PySpark.

Spark automatically stores intermediate data from various shuffle processes. However, it is recommended to use RDD's persist() function. There are many levels of persistence for storing RDDs in memory, disk, or both, with varying levels of replication. The following persistence levels are available in Spark:

MEMORY ONLY: This is the default persistence level and is used to store RDDs on the JVM as deserialized Java objects. In case the RDDs are too large to fit in memory, the partitions are not cached and must be recomputed as needed.

MEMORY AND DISK: On the JVM, RDDs are stored as deserialized Java objects. In case of insufficient memory, partitions that do not fit in memory will be left on disk and data will be read from the drive as needed.

MEMORY ONLY SER: RDD is stored as one byte per partition of serialized Java objects.

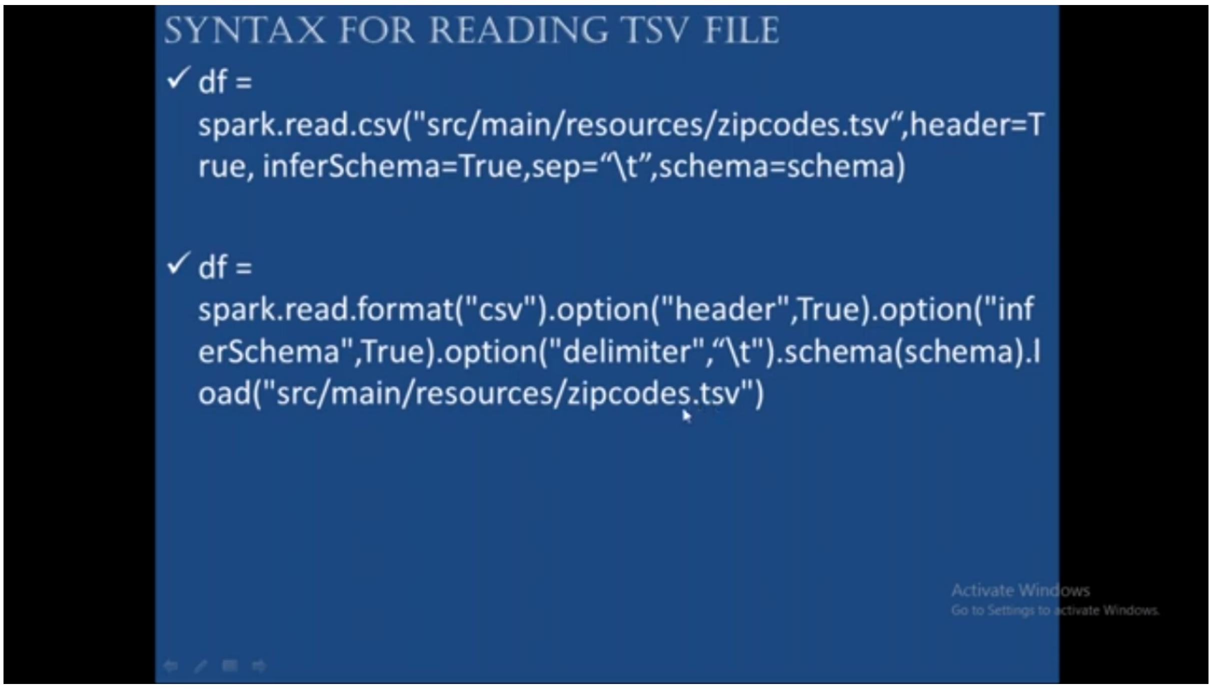
DISK ONLY: RDD partitions are stored on disk only.

OFF HEAP: This level is like MEMORY ONLY SER, except that the data is stored in off-heap memory.

The persist() function has the following syntax for using persistence levels:

```
df.persist(StorageLevel.)
```

Reading csv/tsv or any formatted files



SYNTAX FOR READING TSV FILE

- ✓ df = spark.read.csv("src/main/resources/zipcodes.tsv“,header=True, inferSchema=True,sep="\t”,schema=schema)
- ✓ df = spark.read.format("csv").option("header",True).option("inferSchema",True).option("delimiter",“\t").schema(schema).load("src/main/resources/zipcodes.tsv")